Tags: Database schemas

2005-04-24
Recently, on the del.icio.us mailinglist, I asked the question “Does anyone know the database schema of del.icio.us?”. I got a few private responses so I wanted to share the knowledge with the world.

The Problem: You want to have a database schema where you can tag a bookmark (or a blog post or whatever) with as many **tags** as you want. Later then, you want to run queries to constrain the bookmarks to a **union** or **intersection** of tags. You also want to exclude (say: minus) some tags from the search result.

Apparently there are three different solutions (**Attention: **If you are building a websites that allows users to tag, be sure to have a look at **my performance tests** as performance seems to be a problem on larger scaled sites.)

“MySQLicious” solution

id	url	tags
1	http://laughingmeme.org/archives/002918.html	tags architecture index
2	http://bradchoate.com/weblog/2004/10/06/delicic	del.icio.us tags aggregator ideas
3	http://daryl.learnhouston.com/?p=134	rss wordpress

delicious
id
url
description
extended
tags
date
hash

In this solution, the schema has got just one table, it is **denormalized**

I named this solution “MySQLicious solution” because **MySQLicious** imports del.icio.us data into a table with this structure.

Intersection (AND)

Query for `search+webservice+semweb`:

```
SELECT -

FROM `delicious`

WHERE tags LIKE "%search%"

AND tags LIKE "%webservice%"

AND tags LIKE "%semweb%"
```

Union (OR)

Query for `search|webservice|semweb`

```
SELECT -

FROM `delicious`

WHERE tags LIKE "%search%"

OR tags LIKE "%webservice%"

OR tags LIKE "%semweb%"
```

Minus

Query for `search+webservice-semweb`

```
SELECT -

FROM `delicious`

WHERE tags LIKE "%search%"

AND tags LIKE "%webservice%"

AND tags NOT LIKE "%semweb%"
```

Conclusion

The advantages of this solution:

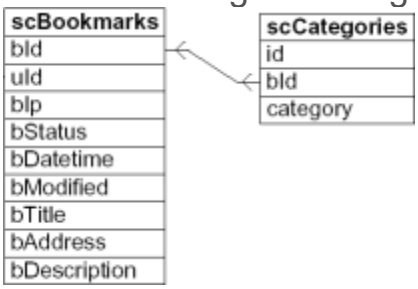
- just one table
- the queries are very straightforward
- one can also achieve results via fulltextsearch. That might be a little faster.
- queries are quite slow according to some commenters. Fulltext search would speed up a bit. I **did some performance tests** to prove that.
- **In my follow up post I dealt with MySQL fulltext concerning tagging**.

Disadvantages:

- You have a limit on the number of tags per bookmark. Normally you use a 256byte field in your DB (`VARCHAR`). Otherwise, if you took a `text` field or similar, the query times would slow down, I suppose
- Patrice **noticed** that `LIKE "%search"` will also find tags with “websearch”. If you alter the query to `LIKE " %search% "` you end up having a messy solution: You have to add a space to the beginning of the tags value to make this work.

“Scuttle” solution

Scuttle organizes its data in two tables. That table “scCategories” is the “tag”-table and has got a foreign key to the “bookmark”-table.



Intersection (AND)

Query for `bookmark+webservice+semweb`:

```
SELECT b.*

FROM scBookmarks b, scCategories c

WHERE c.bId = b.bId

AND (c.category IN ('bookmark', 'webservice', 'semweb'))

GROUP BY b.bId

HAVING COUNT( b.bId )=3
```

First, all bookmark-tag combinations are searched, where the tag is “bookmark”, “webservice” or “semweb” (`c.category IN ('bookmark', 'webservice', 'semweb')`), then just the bookmarks that have got all three tags searched for are taken into account (`HAVING COUNT(b.bId)=3`).

Union (OR)

Query for `bookmark|webservice|semweb`:

Just leave out the `HAVING` clause and you have union:

```
SELECT b.*

FROM scBookmarks b, scCategories c

WHERE c.bId = b.bId

AND (c.category IN ('bookmark', 'webservice', 'semweb'))

GROUP BY b.bId
```

Minus (Exclusion)

Query for `bookmark+webservice-semweb`, that is: bookmark AND webservice AND NOT semweb.

```
SELECT b. *
```

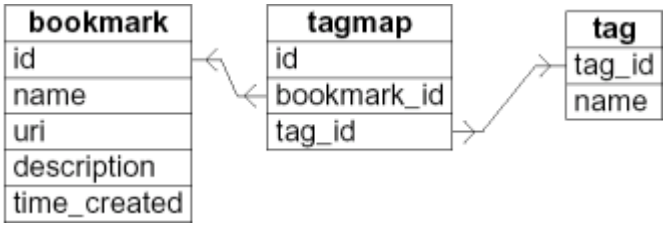
```
FROM scBookmarks b, scCategories c

WHERE b.bId = c.bId

AND (c.category IN ('bookmark', 'webservice'))

AND b.bId NOT

IN (SELECT b.bId FROM scBookmarks b, scCategories c WHERE b.bId = c.bId AND c.categor
y = 'semweb')

GROUP BY b.bId

HAVING COUNT( b.bId ) =2
```

Leaving out the `HAVING COUNT` leads to the Query for “bookmark|webservice-semweb”.
Credits go to **Rhomboid** for **helping me out with this query**.

Conclusion

I guess the main advantage of this solution is that it is more normalized than the first solution, and that you can have unlimited number of tags per bookmark.

“Toxi” solution



Toxi came up with a three-table structure. Via the table “tagmap” the bookmarks and the tags are n-to-m related. Each tag can be used together with different bookmarks and vice versa. This DB-schema is also used by **wordpress**.

The queries are quite the same as in the “scuttle” solution.

Intersection (AND)

Query for `bookmark+webservice+semweb`

```
SELECT b.*

FROM tagmap bt, bookmark b, tag t

WHERE bt.tag_id = t.tag_id

AND (t.name IN ('bookmark', 'webservice', 'semweb'))

AND b.id = bt.bookmark_id

GROUP BY b.id

HAVING COUNT( b.id )=3
```

Union (OR)

Query for `bookmark|webservice|semweb`

```
SELECT b.*

FROM tagmap bt, bookmark b, tag t

WHERE bt.tag_id = t.tag_id

AND (t.name IN ('bookmark', 'webservice', 'semweb'))

AND b.id = bt.bookmark_id

GROUP BY b.id
```

Minus (Exclusion)

Query for `bookmark+webservice-semweb`, that is: bookmark AND webservice AND NOT semweb.

```
SELECT b. *

FROM bookmark b, tagmap bt, tag t

WHERE b.id = bt.bookmark_id

AND bt.tag_id = t.tag_id

AND (t.name IN ('Programming', 'Algorithms'))

AND b.id NOT IN (SELECT b.id FROM bookmark b, tagmap bt, tag t WHERE b.id = bt.bookma
rk_id AND bt.tag_id = t.tag_id AND t.name = 'Python')

GROUP BY b.id

HAVING COUNT( b.id ) =2
```

Leaving out the `HAVING COUNT` leads to the Query for `bookmark|webservice-semweb`.
Credits go to **Rhomboid** for **helping me out with this query**.

Conclusion

The advantages of this solution:

- You can save extra information on each tag (description, tag hierarchy, …)
- This is the most normalized solution (that is, if you go for **3NF**: take this one :-)

Disadvantages:

- When altering or deleting bookmarks you can end up with tag-orphans.

If you want to have more complicated queries like (bookmarks OR bookmark) AND (webservice or WS) AND NOT (semweb or semanticweb) the queries tend to become very complicated. In these cases I suggest the following query/computation process:

1. Run a query for each tag appearing in your “tag-query”:

```
2. SELECT b.id FROM tagmap bt, bookmark b, tag t

3. WHERE bt.tag_id = t.tag_id AND b.id = bt.bookmark_id AND t.name = "semweb"
```

4. Put each id-set from the result into an array (that is: in your favourite coding language). You could cache this arrays if you want..
5. Constrain the arrays with union or intersection or whatever.

In this way, you can also do queries like `(del.icio.us|delicious)+(semweb|semantic_web)-search`. This type of queries (that is: the brackets) cannot be done by using the denormalized "MySQLicious solution".
This is the most flexible data structure and I guess it should scale pretty good (that is: if you do some caching).

Update May, 2006. This arcticle got quite some attention. I wasn't really prepared for that! It seems people keep referring to it and even some new sites that allow tagging give credit to my articles. I think the real credit goes to the contributers of the different schemas: **MySQLicious**, **scuttle**, **Toxi** and to all the contributors of the comments (be sure to read them!)

P.S. Thanks to **Toxi** for sending me the queries for the three-table-schema, Benjamin Reitzammer for pointing me to **a loughing meme article** (a good reference for tag queries) and powerlinux for pointing me to **scuttle**.

Tagsystems: performance tests

2005-06-19

In my **previous article named "Tags: database schemas"** we analysed different database schemas on how they could meet the needs of tag systems. In this article, the focus is on performance (speed). That is: if you want to build a tagsystem that performs good with about 1 million items (bookmarks for instance), then you may want to have a look at the following result of my performance tests.
In this article I tested tagging of bookmarks, but as you can tag pretty much anything, this goes for tagging systems in general.

I tested the following schemas (I keep the naming from the previous article):

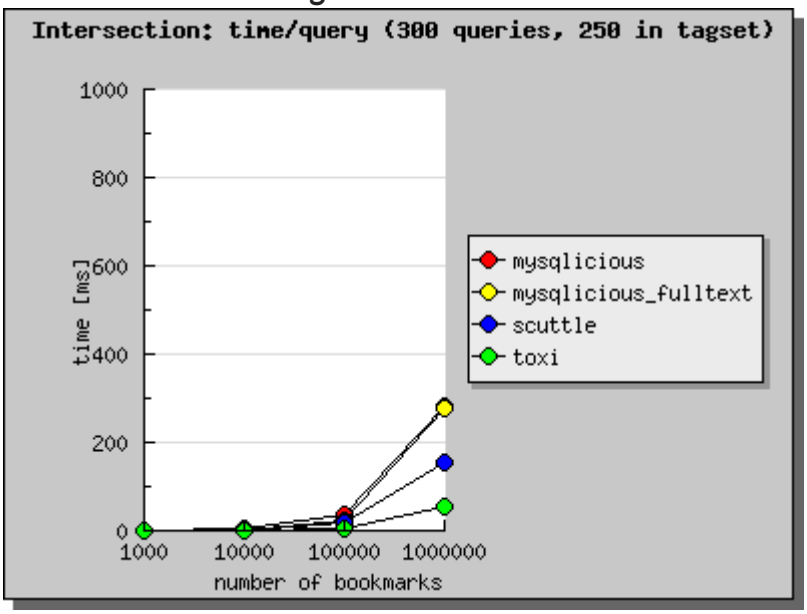
- **mysqlicious**: One table. Tags are space separated in column "tags"; **as introduced**
- **mysqlicious fulltext**: Same schema but with **mysql fulltext** on the tag column; **as introduced**
- **scuttle**: Two tables: One for bookmarks, one for tags. Tag-table has foreign key to bookmark table; **as introduced**
- **toxi**: Three tables: One for bookmarks, one for tags, one for junction; **as introduced**

You may want to have a close watch at the details of the schemas when having a look at the **sql-create-table-queries**.

But let's go directly to the results. The x-axis depicts the number of bookmarks in the corresponding database, on the y-axis you see how much time each query took to execute.

Results

Intersection: 250 tag set



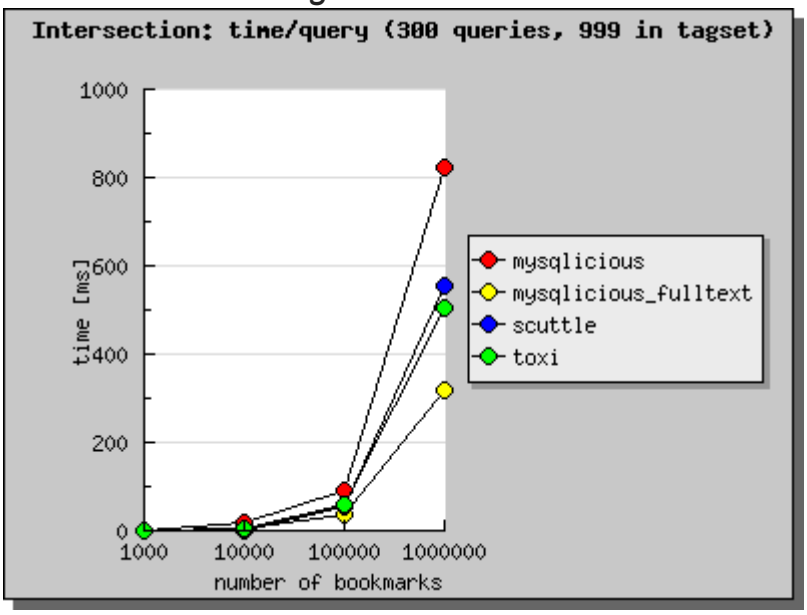
The first two tests are done with 250 tags in the small dataset. I think the queries in the “1 million bookmarks database” are the only size we should pay attention to. I mean if you have a small number of bookmarks, performance isn’t really a thing to bother..

We run intersection queries, like

I want to search for bookmarks tagged with "design" and "html"

You see that, not surprisingly, mysqlicious with its `WHERE tag LIKE "% tag %"` is very slow. That is, MySQL has to go through the whole dataset and test each bookmark against the query.
What actually **is** surprising me, is that the fulltext search of mysql is not that high-performance. In fact it is not faster then the `LIKE`-query in the MySQLicious DB. This really disappointed me. I tried to do any quirks possible to make this faster as **I think, a tag-database-system with mysql fulltext would be very easy and like the only thing you should head to..**.
What is surprising me too, is that the queries on the 3 table schema are about double as fast the the ones on the two-table ones(**take a look at the queries** if you think you could give me a hint on this). Noticeable is, that in the scuttle and toxi-variant, the more queries were run, the faster they were. I didn’t do any tests with queries and inserts mixed so this may be coming from just plain good caching and this effect possible doesn’t show up on live bookmark management systems.

Intersection: 999 tag set



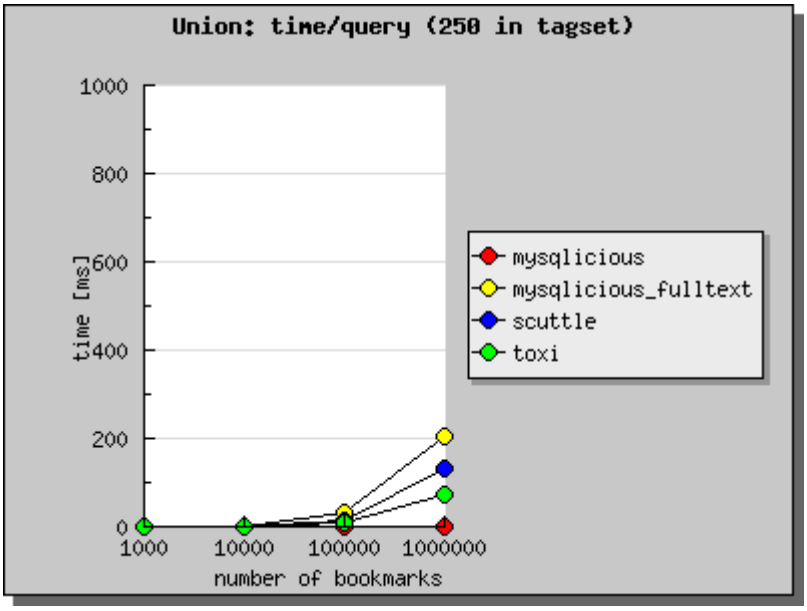
Now have a look to what happens if we broaden our small tag set: MySQLicious with fulltext suddenly gets the performance leader. That means, if you have a bookmark management system with diverse tags (this most probably comes from the fact that there are many users), the fulltext solution is possibly the way to go.
So now, as you see, choosing the right schema is all about tag distribution. In my previous post about guessing the overall tag distribution on **del.icio.us**, I came to the conclusion, that delicious' most popular tag "design" is showing up in 3.2% of all bookmarks on **del.icio.us**. So then, what is the mean tag distribution?

- If we say 1% (a tag shows up in 1/100 of all bookmarks on an average) that makes our small tag set 250 tags big
- If we say 0.25%, the small tag set grows to a size of 1000
- If we say 0.1%, the small set will contain 2500 tags

So I'd suggest that if your average distribution is 1%, take "toxi", if the distribution is broader, take "MySQLicious fulltext".

If you take a closer look, you can see that the fulltext schema stayed as fast as when queried in the 250 tag set. That means, if you want to go sure your tag system responds ok in every situation, you should go with the "mysql fulltext" schema.

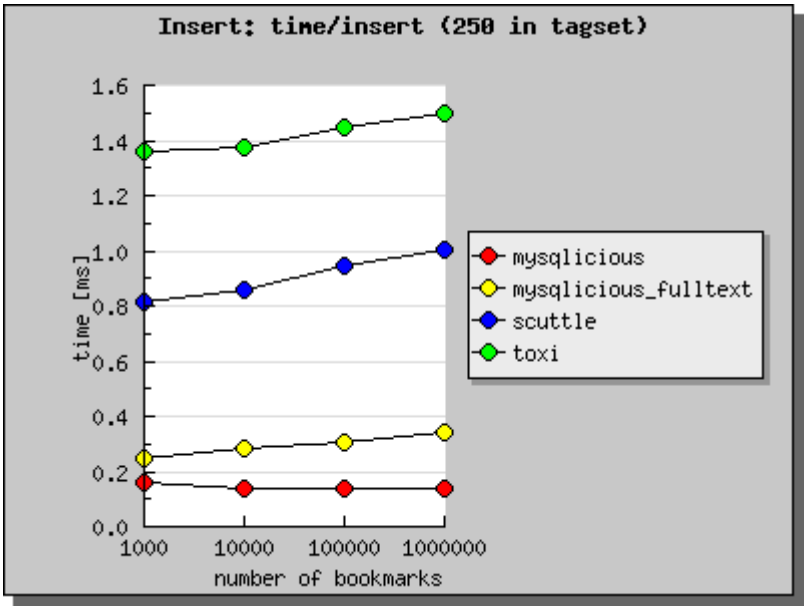
Union



When doing a union query we say

I want to search for all the bookmarks that are tagged either with “delicious” or “del.icio.us”
This queries, you guessed, are handled the fastest by “MySQLicious schema” with its `LIKE`-queries: MySQL seeks through the bookmarks, harvesting all bookmarks with one of the given tags and says “I’m finished!” when it was at bookmark number #968, because it found 50 bookmarks. Whereas in the other schemas, MySQL has to join the tags with the bookmarks first and only then could search though it..

Insert



When comparing the different schemas on the time of the insert-“statements” of one bookmark, the result isn’t very surprising (notice that I’ve changed the scale of the y-axis).
Mysqlicious with it’s 1 table is very fast indeed, its variation with fulltext had to create the fulltext index and therefore is a bit slower. Scuttle, with its 2 tables and toxi with its 3 tables are at least two respectively three times as slow. I have to remark, that I used quite a bit of caching for the toxi schema, as I didn’t want hours to have the data ready..

I guess it doesn’t really make sense to base your decision, which schema to take on the time for an insert: Bookmark inserts are about 100 times as fast as the intersection queries..

«What? That slow??»
You said it. You don’t want your intersection queries take 0.2 seconds each. That would bring your system to its knees.

There are some recipes to avoid that:

Caching

I think, you don’t come around good old caching. I think that you could cache results to a query like “mysql+tagging” for about an hour or so. If a user queries his own items, I would lower the cache time (as up-to-dateness is

more important with his own items).
Then, I expect if you for instance cache items per tag and intersection them with a decent algorithm, that could be faster..

The Best Of Both Worlds

I think you could have “mysqlicous fulltext” and “toxi” running at the same time. That means you have to update/insert in both schemas but when you have to query, you could take the one you think is faster: For simple union the mysqlicious without fulltext search, for intersection queries with common tags the toxi, and for those with uncommon tags the mysqlicious fulltext variant.

Slicing and dicing

You could “slice and dice” data, that is: you slice your user/tag/item-room and build fact tables. You “prebuild” your results in a way. This way, inserts take long but queries themself should be much faster. In our examples, you would for instance first query the tag-intersections on “toxi” and then get the facts about each bookmark out of the “mysqlicious”-fact-table. But you really should read Nitins posts, as they give a lot of insight.

Using a non RDBMS system

Update: It’s been about a year since I wrote that article, and during that year I came to the conclusion that **RDBMS** systems don’t scale good in systems that have more than 1 million items. Yes, this is a warning: If you are planning to build a large scale system then look for alternatives to **RDBMS** systems. To quote Joshua Schachter, founder of **delicious**:
«tags doesn’t map to sql at all. so use partial indexing.»[**Joshua Schachter at Carson Summit**]
I didn’t try any of the non-RDBMS system but it looks like **Apache Lucene** and **Hadoop**.

Performance Tests Setup

Now, if you have read that far, you probably want to know some background information: As you noticed, for each schema, I set up 4 databases, one database holding 1000 bookmarks, the next 10’000, then 100’000 and the

fourth 1 million bookmarks. The inserted tags (as well as urls) are random English words taken from two sets of tags:

- the large set containing about 44’000 tags (that are simple English words)
- the small set is varying in size (the results shown here are taken from 250 and 999 tag sets)

Every bookmark gets one to ten tags attached. Every odd tag is from the large set, alternately taken from small and large set. Every schema got exactly the same bookmarks and tag data.

Then every schema got queried with an alternately 1-3 tag query. So the first query is for instance just “blog”, the second “design+css”, the third “webdesign+music+software”, the fourth again with just one tag an so forth..
All the tags for the queries are taken from the small set so that the queries don’t all end in empty results..
All the queries are tested and work. The outcome of each query on the three schemas is exactly the same.

Mysql Setup
I used mysql 4.0.21.
An excerpt from `/etc/my.cnf` (I think these are the relevant settings to this performance test)

```
key_buffer=300M

query_cache_size=30M

query_cache_limit=30M

table_cache = 64

ft_min_word_len = 2

ft_stopword_file = ''
```

System
CPU: 3GHz Dual Xeon Cache: 1MB Harddisk: SCSI Ultra 320 Atlas 10K, no RAID RAM: 3GB

Assumptions

- Queries select just the id of a bookmark. I assume that you have to do a second query to get all the wished data to display. I know that this is not fair towards the mysqlicious schema.
- I left out user data, as I assume, user data columns wouldn't change the outcome of this tests. I wanted to keep the schemas as simple as possible.
- Each query is done with `LIMIT 50` as I assume that a normal application doesn't want to get all bookmarks. I assume nobody wants to `order` bookmarks by any dimension, because this would be **very** expensive (ever wondered why you cannot sort bookmarks on **del.icio.us** by date or similar? You get it..)

Acknowledgements

Thanks to **Citrin**, the company I work, to let me use our new server to run the queries. The server didn't have much anything else to do so the results should be accurate.
The graphs are done using **JpGraph**. Very easy to use and produces beautiful images.